Starting with R

R is an integrated suite of software facilities for data manipulation, calculation, statistics and graphical display.

R is extremely useful to deal with structured data of the following format: each line is a data point and each column contains some information about that data point (you can think of an Excel spreadsheet). Typically, annotated data or results from experiments will have this format.

In this unit, we will give you a preliminary taste of R, and hopefully make you want to learn more about it. We will look at the English dative alternation. In English, you can either say *The teacher gave the toys to the children* using an object and a prepositional phrase (henceforth NP_PP), or you can say *The teacher gave the children the toys* using two objects (henceforth the NP_NP construction). For the examples above, the thing that gets given (the toys) is referred to as the *theme* and the children is the *recipient*.

Bresnan et al. (2007) proposed a model of the factors that affect the choice of dative construction for adult data: length of the theme and recipient, animacy of these, lexical expression, etc. Here we are going to look at a database of dative constructions by children. The goal of the project was to identify whether children are sensitive to the same factors as adults. We will start by investigating the child database *child_dative.csv*, which you can download from Carmen. The database contains both dative constructions uttered by children as well as by the adult caretakers of these children. You can read more about this study in Marie-Catherine de Marneffe, Scott Grimm, Inbal Arnon, Susannah Kirby and Joan Bresnan. 2012. "A statistical model of grammatical choices in child production of datives sentences". Language and Cognitive Processes 27(1):25-61.

A good starting point for the material we are going to cover is the first chapter of Harald R. Baayen. (2008). "Analyzing Linguistic Data. A Practical Introduction to Statistics Using R." Cambridge University Press.

Basics

Launch R. We get a R console in which we will be able to type. The R prompt is ">".

We will want to load data from our computer, so we need to know where we are. By default, the *working directory* is the user home directory. The function *getwd()* gives you the current working directory. So on my laptop this is what I get:

```
> getwd()
[1] "/Users/mcdm"
```

On the computer lab machines, you will get:

```
> getwd()
[1] "Users/buckeye"
```

If you want, you can change the working directory. Go to "Misc" > "Change working directory". This will open a file system browser which allows you to choose the folder you want as you working directory (e.g., the directory where the files you will want to load in R live).

As with Python, we can just use R as a calculator.

```
> 5 + 3 * 2
[1] 11

> sqrt(81)
[1] 9
```

Again, remember that people are lazy. If you type the beginning of a command and hit the "tab" key, R will fill it for you. It is also very useful when you don't fully remember the exact command but are pretty sure of how it starts!

We can have *variables* to which we assign a *value*.

```
> x = 5
> x
[1] 5
```

On top of the = sign, R also allows left-arrow and right-arrow as assignment operator.

```
> x <- 5 + 2
> x
[1] 7

> 5 + 5 -> y
> y
[1] 10
```

Reading and accessing data

But we don't really care about calculators ;-) What we want is to explore data. The child dative database mentioned above is a CSV file (comma-separated values). Each line has the same number of fields, separated by a comma. The easiest way to deal with files in R is to have them under this "CSV" format. If you have an Excel table, you can save it as a CSV file. R has a function *read.csv()* that allows us to load such files. The function takes one obligatory **argument**: the file you want to load. You need to give the path to that file *relative to* the working directory.

```
> read.csv("Downloads/child_dative.csv")
```

What happens when I do this? The content of the file gets printed to the console, but that's it. This isn't very useful. We will want to *do* stuff with that data. We need to *store* it. So we will choose a **variable** name and assign to it the output of the read.csv function.

```
> dative = read.csv("Downloads/child_dative.csv")
```

We can look in the manual to have a better idea of what *read.csv* is doing.

```
> help(read.csv)

Description

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

Usage

read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", row.names, col.names,
           as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text)

read.csv(file, header = TRUE, sep = ",", quote="\"", dec=".",
         fill = TRUE, comment.char="", ...)

...
```

We can see that by default it will assume a HEADER row (meaning that the first line of the file will contains the names of the fields) and that the separator is a comma.

It is good practice to always make sure that everything is loaded properly. Take a peek at the start of the dataset.

```
> head(dative)

  PID   File     Age Age.numeric Age_group Line Group Child Speaker
1  26 abe064  3;0.29        3.08         2   NA child   abe     abe
2  27 abe064  3;0.29        3.08         2   NA child   abe     abe
3  41 abe109  3;6.13        3.54         2   NA child   abe     abe
4  42 abe109  3;6.13        3.54         2   NA child   abe     abe
5 546 abe142 3;10.14        4.30         3   NA child   abe     abe
6  45 abe112  3;6.22        3.56         2   NA child   abe     abe
                                                                                                         Sentence Prime
1                                                                                   " hey # come give me a hug ."    NP
2                                                                                     " give me a hug # please ."    NP
3                                                                               I'll show you the little bottle .    NP
4 I'm gon (t)a show you the little bottle over across the street do you wan(t) (t)a come over across the street ?    NP
5                                                                                   can I show you this one now ?    NP
6                                                                                            give it back to me .    PP
                            Prime_line Repetition SameVerb Prime_distance Prev_dative_constr Prime_Speaker Prime_verb Verb
1                oh come give me a hug          1        1             10                 NP           FAT       give give
2             hey # come give me a hug          1        1              6                 NP           FAT       give give
3      I'll show you the little bottle          1        1              2                 NP           CHI       show show
4      I'll show you the little bottle          1        1              2                 NP           CHI       show show
5 ok I wan(t) (t)a show you this one .          1        1              5                 NP           CHI       show show
6                   give it back to me          1        1              1                 PP           CHI       give give
                                     Theme Recipient Theme.givenness Rec.givenness Theme.animacy Rec.animacy Theme.toy.animacy Rec.toy.animacy
1                                    a hug        me           given         given             0           1                0               1
2                                    a hug        me           given         given             0           1                0               1
3                        the little bottle       you           given         given             0           1                0               1
4 the little bottle over across the street       you           given         given             0           1                0               1
5                                 this one       you             new         given             0           1                0               1
6                                       it     to me           given         given             0           1                0               1
  Theme.pron Rec.pron Theme.length Rec.length Construction
1    lexical  pronoun            2          1           NP
2    lexical  pronoun            2          1           NP
3    lexical  pronoun            3          1           NP
4    lexical  pronoun            7          1           NP
5    pronoun  pronoun            2          1           NP
6    pronoun  pronoun            1          1           PP
```

You can also look at the end of the dataset:

```
> tail(dative)
```

In most cases, the data you load is a dataset you are familiar with (probably built by you!), so you probably know how much lines this dataset contains. You can check that with the *nrow* function. This function takes one argument: the object you want to count lines of.

```
> nrow(dative)
[1] 1353
```

Remember what we did for Python. We could check the type of objects. We can do the same here using the "class" function.

```
> class(dative)
[1] "data.frame"
```

You can think of a data frame as a matrix. Think of a Battleship game! We can easily access a cell by specifying the row and the column.

Let's first try this on a toy example which we will create from scratch. We will create 2 vectors and put them together to create a data frame:

```
> names = c("Marie", "Micha", "Olivier")
> ages = c(35, 31, 32)
> d = data.frame(names, ages)

> d
  names         ages
1 Marie         35
2 Micha         31
3 Olivier       32
```

By definition a data frame contains vectors of same length. What happens if we do this?

```
> ages = c(35, 31, 32, 4)
> d = data.frame(names, ages)
```

We get an error message telling us that we are trying to construct a data frame with vectors of different lengths. Build the data frame properly again.

Now we can access the age of Marie for example. It is in cell (1,2):

```
> d[1,2]
```

We can also access a whole row or a whole column:

```
> d[1,]                    # getting the first row
> d[,2]                    # getting the second column
```

Going back at getting Marie's age: more realistically we will not know in which cell it is. But we will know that we want to get the age corresponding to the value "Marie" for names. We specify a column using the $ sign:

```
> d$names
```

So to get the age of Marie, we will specify which row we want to look at, and specify which column we want in that row. Let's do this step by step. What does the following get us?

```
> d[d$names == "Marie",]
```

Now we specify the column we want:

```
> d[d$names == "Marie",]$ages
```

How do you get the names of people below 35?

We can also easily add another column to the data frame:

```
> genders = c("f", "m", "m")
> d = cbind(d, genders)
```

And now, how do we take the subset of men only?

```
> men = d[d$genders == "m",]
```

We can also use the *subset* function to do this. We will specify the condition which we want the data to satisfy. Rows satisfying the condition will be kept.

```
> men2 = subset(d, genders == "m")
```

Two other useful commands are the *names* and *levels* ones. Look at the manual to see what they do and try them on our toy dataset we just made.

What happens if we try the *levels* function on the *genders* column in our original data frame? And what about the *men* subset?

Note that *levels* is defined for a categorical variable only.

```
> levels(men$genders)
[1] "f" "m"
```

That's something worth knowing about R. Unless you explicitly told R to drop the levels when subsetting the data, it keeps the original levels. To drop the levels, do:

```
> droplevels(men)
> men = droplevels(men)
```

Now try again the *levels* function. Looks better, right?

```
> levels(men$genders)
[1] "m"
```

Data exploration

Let's go back to our dative data. A method that can be quite useful to look at what we have in the data is the *summary* one:

```
> summary(dative)

           PID                  File            Age         Age.numeric      Age_group          Line
 Min.   :   1.0   nina05.cha:  26   2;11.28:  32   Min.   :1.900   Min.   :1.000   Min.   :  14.0
 1st Qu.: 209.0   nina46.cha:  21   2;6.17 :  30   1st Qu.:2.490   1st Qu.:1.000   1st Qu.: 693.5
 Median : 412.0   nina48.cha:  20   2;3.18 :  28   Median :2.950   Median :1.000   Median :1475.0
 Mean   : 483.4   adam08.cha:  19   2;0.10 :  26   Mean   :3.040   Mean   :1.596   Mean   :1516.5
 3rd Qu.: 718.0   nina04.cha:  19   3;1.4  :  26   3rd Qu.:3.250   3rd Qu.:2.000   3rd Qu.:2253.0
 Max.   :1244.0   nina07.cha:  18   4;2.17 :  25   Max.   :5.383   Max.   :3.000   Max.   :4385.0
                  (Other)   :1230   (Other):1186                                   NA's   :530
   Group        Child        Speaker                                     Sentence                Prime
 cds  :823   abe   : 75   *MOT   :629   who \002\005(1)\002\006gave it to you ?           :   7   0 :894
 child:530   adam  :429   adam   :221   give me ride .                                    :   5   NP:356
             naomi : 21   nina   :146   who \002\005(1)\002\006gave them to you ?         :   5   PP:103
             nina  :613   *INV   :124   give me it .                                      :   4
             sarah : 19   abe    : 75   \002\005(1)\002\006give it to me .                :   3
             shem  :163   *URS   : 35   come here # and \002\005(1)\002\006give it to me .:   3
             trevor: 33   (Other):123   (Other)                                           :1326
                     Prime_line     Repetition         SameVerb       Prime_distance   Prev_dative_constr
                          :1147   Min.   :0.00000   Min.   :0.0000   none   :561                :875
 ??? Couldn't find it     :   3   1st Qu.:0.00000   1st Qu.:1.0000          :328    NP          :326
 give me ride             :   3   Median :0.00000   Median :1.0000   1      :154    PP          : 88
 give her milk .          :   2   Mean   :0.03769   Mean   :0.9109   2      :100    NP question: 24
 give me it .             :   2   3rd Qu.:0.00000   3rd Qu.:1.0000   3      : 50    imperative :  7
 give me that broken one !:   2   Max.   :1.00000   Max.   :1.0000   4      : 45    NP relative:  7
 (Other)                  : 194                     NA's   :1151     (Other):115    (Other)    : 26
 Prime_Speaker   Prime_verb    Verb           Theme        Recipient    Theme.givenness Rec.givenness
        :880           :879   give:1095   it       : 176   me      :323   given:763       given:1144
 *MOT   :149   give    :393   show: 258   them     :  34   you     :269   new  :590       new  : 209
 *CHI   : 72   show    : 65               something:  28   him     : 90
 CHI    : 65   bring   :  4               that     :  25   her     : 86
 CH     : 41   send    :  3               some     :  22   to me   : 59
 MO     : 38   tell    :  3               one      :  21   to you  : 51
 (Other):108   (Other):  6               (Other)  :1047   (Other):475
 Theme.animacy      Rec.animacy     Theme.toy.animacy Rec.toy.animacy    Theme.pron      Rec.pron
 Min.   :0.00000   Min.   :0.0000   Min.   :0.00000   Min.   :0.0000   lexical:1045   lexical: 351
 1st Qu.:0.00000   1st Qu.:1.0000   1st Qu.:0.00000   1st Qu.:1.0000   pronoun: 308   pronoun:1002
 Median :0.00000   Median :1.0000   Median :0.00000   Median :1.0000
 Mean   :0.02217   Mean   :0.9076   Mean   :0.06726   Mean   :0.9815
 3rd Qu.:0.00000   3rd Qu.:1.0000   3rd Qu.:0.00000   3rd Qu.:1.0000
 Max.   :1.00000   Max.   :1.0000   Max.   :1.00000   Max.   :1.0000
  Theme.length     Rec.length     Construction
 Min.   : 1.00   Min.   :1.000   NP:1014
 1st Qu.: 1.00   1st Qu.:1.000   PP: 339
 Median : 2.00   Median :1.000
 Mean   : 2.18   Mean   :1.113
 3rd Qu.: 2.00   3rd Qu.:1.000
 Max.   :15.00   Max.   :7.000
```

What is great about this command is that we see how the values of the variables are interpreted by R. This is very important. Let's look at what is happening with the *animacy* variable. Right now it is seen as a *numeric* variable. How is it coded? Let's look at it:

```
> dative$Theme.animacy
```

It is coded as a binary variable with 0 and 1. But the 0s and the 1s are interpreted as integers right now, which is not good. This is actually a *categorical* variable: we could have coded it with "animate" and "inanimate" for example, instead of "1" and "0". We can force R to see it as such:

```
> dative$Theme.animacy = as.factor(dative$Theme.animacy)
```

We will do it for the recipients too:

```
> dative$Rec.animacy = as.factor(dative$Rec.animacy)

> class(dative$Theme.animacy)
[1] "factor"
```

Re-run the *summary* function on the *dative* data frame. What is different now?

Assume that this 0 and 1 coding is confusing you. You can easily change that. There are multiple ways to do this. Here is one: we will add a new column to our data frame, and put in it the values we want ("animate" if the Theme.animacy was 1, "inanimate" if the Theme.animacy was 0):

```
# create a new column with "animate" as value
> dative$Theme.animacy2 = "animate"

# where Theme.animacy was 0, we change the "animate" value to "inanimate"
> dative[dative$Theme.animacy == "0",]$Theme.animacy2 = "inanimate"

# make sure we have a categorical variable
> dative$Theme.animacy2 = as.factor(dative$Theme.animacy2)
```

Now let's look into more details what is happening in the dative constructions uttered by children only. To simplify the expression of the commands, we will create a subset containing only the data for the children.

```
> child = subset(dative, Group == "child")
```

Now create a subset with the child-directed speech data. How many dative constructions do we have per subsets?

```
> nrow(child)
[1] 530
```

Here are some questions that we might ask ourselves:

1. What are the verbs in this database?

This information was already in the output of the *summary* method. We can also use the *levels* function. How?

2. How often do we have pronominal recipients in the different constructions, for the adults and for the children?

To answer this, we will create what is called a *contingency table* or *cross tabulation*: it just displays the frequency distribution of some variables. This is done with the *xtabs* function in R:

```
> xtabs(~ Rec.pron + Construction, data = child)
         Construction
Rec.pron   NP  PP
  lexical  50  56
  pronoun 358  66

> xtabs(~ Rec.pron + Construction, data = cds)
         Construction
Rec.pron   NP  PP
  lexical 139 106
  pronoun 467 111
```

3. When we did some statistical modeling to see which factors influenced children's construction choice, *animacy* didn't turn out to be significant. Look at the data and try to see why this actually makes sense. We want to get the frequency distribution of the animacy of the recipient per construction.
4. We can also look at how given and new themes are expressed (pronouns or lexical noun phrases). Let's look at this for children as well as for adults:

```
> xtabs(~ Theme.pron + Theme.givenness, data = child)
          Theme.givenness
Theme.pron given new
   lexical   215 182
   pronoun   114  19

> xtabs(~ Theme.pron + Theme.givenness, data = cds)
                 Theme.givenness
Theme.pron given new
            lexical   266 382
            pronoun   168   7
```

It seems quite different. We can easily do a chi-square test on this :-)

Data visualization

We might also want to visualize the data with graphs. Just to show you some of the basic plotting functions in R, we can try the following.

Make a bar plot of the lengths of the themes in the child data:

```
> barplot(xtabs( ~ child$Theme.length))
```

We can specify x-axis and y-axis labels if we want:

```
> barplot(xtabs( ~ child$Theme.length), xlab = "Number of words in theme", ylab = "Number of utterances")
```

How do we add a global title for the graph? Look in the manual:

```
> help(barplot)
```

Let's say we want to look together at a barplot of the theme lengths for the children and one for the adults. We can define how many panels we want using *par*. We will set the parameter *mfrow* to the matrix of panels we want:

```
> par(mfrow = c(1,2))                          # this create a 1 x 2 matrix
> barplot(xtabs( ~ child$Theme.length))
> barplot(xtabs( ~ cds$Theme.length))
```

Perhaps it would be more insightful to see how the length of the theme patterns per construction:

```
> barplot(xtabs(~ child$Construction + child$Theme.length))
```

Mmh, we might want to add a legend here. I never remember how to do this, but I just look in the manual!

```
> barplot(xtabs(~ child$Construction + child$Theme.length), legend.text = TRUE)
```

I don't really like stacked bars... This isn't that helpful visually. Can we juxtapose them perhaps? Look in the manual! Add labels to the graph too.